Universidad Nacional de Ingeniería Facultad de Ciencias

Implementation and Evaluation of a Distributed Agent System for Concurrent Task Management

Espinoza Pari, Franklin , Cleber Aguado Gutierrez

Joel Chavez Chico

Profesor

Yuri Nuñez Medrano

Julio, 2024

1. Introduction

This report details the development and implementation of a distributed agent system designed to process tasks concurrently, in parallel, and in both homogeneous and heterogeneous distributed environments. These distributed agents operate as independent threads on distributed processors and can autonomously fulfill predefined goals and tasks

.

2. Objectives

- Develop distributed agents capable of processing tasks in a parallel and concurrent manner.
- Implement a worker node system that can include both desktop and mobile devices.
- Ensure fault tolerance by automatically redistributing tasks if a node fails.
- Deploy and evaluate the system in LAN and Wi-Fi network environments.
- Measure the system's performance by executing large-scale tasks.

3. System Description

a. Worker Nodes

i. **Desktop Workers:** Fixed nodes that process tasks using an HTTP server and a distributed module.

ii. **Mobile Workers:** Mobile nodes that also use an HTTP server and a distributed module.

b. Agent Tasks

i. **Word Counter:** Counts a word, multiple words, or all words in a text file.

Input:

- n: Number of words
- s_1, s_2, ..., s_n: Words to count

Output:

- freq[s1], freq[s2], ..., freq[sn]: Frequencies of each word

ii. **Keyword Finder:** Searches for specific keywords in the text.

Input:

- n: Number of keywords
- s_1, s_2, ..., s_n: Keywords to search

Output:

- ans_i = {YES, NO}: Whether each keyword exists

iii. **Repeated Keyword Finder:** Identifies keywords that repeat a specified number of times.

Input:

- n: Number of keywords
- r: Frequency of repetition
- s_1, s_2, ..., s_n: Keywords to search

Output:

- m: Number of keywords that repeat r times
- s_1, ..., s_m: Keywords that meet the repetition criteria

4. **Task Distribution Mechanism**
   - Clients (desktop or mobile) send text files along with tasks to the worker nodes.
   - Worker nodes process these tasks and return the results to the clients.
   - If a node fails, the task is redistributed to another node to ensure task completion

5. **Consensus Algorithm and Leader Election**

In our distributed system, achieving consensus among nodes is critical to ensure data consistency and fault tolerance. The consensus algorithm ensures that all nodes agree on a common value or decision even in the presence of node failures or network partitions. The following steps outline the basic consensus algorithm implemented in our system:

Initialization:

- Each node starts in an uncommitted state.
- Nodes are aware of each other and can communicate directly.

Proposal Phase:

- One node, designated as the leader, proposes a value (e.g., a task assignment) to all other nodes.

Voting Phase:

- Nodes receive the proposal and vote to accept or reject it.
- Votes are sent back to the leader.

Decision Phase:

- The leader collects votes. If a majority of nodes accept the proposal, it is considered accepted.
- The leader broadcasts the decision to all nodes.

Commitment Phase:

- Nodes commit to the agreed value and update their state accordingly.
- This ensures consistency across the system.

Fault Tolerance:

- If the leader fails, nodes detect the failure and elect a new leader to continue the consensus process.

6. System Architecture

a. Main Components

- **Clients:** Devices that send tasks to the worker nodes.
- **Worker Nodes:** Devices that process the tasks sent by the clients.
- **Worker Leader:** Central node that coordinates tasks among worker nodes and ensures consistency.

b. **Connectivity**

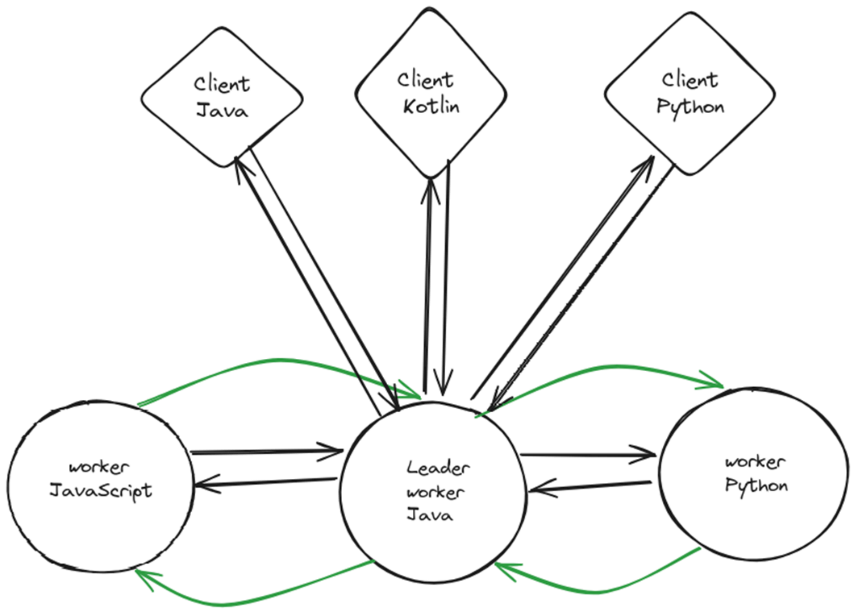
- **Clients and workers:** Communication through the worker leader using low-level sockets.
- **Task redistribution:** Implementation of mechanisms for task transfer in case of node failure.

c. **Operating System**

Worker nodes operate on different operating systems to ensure heterogeneity (SO1 and SO2).

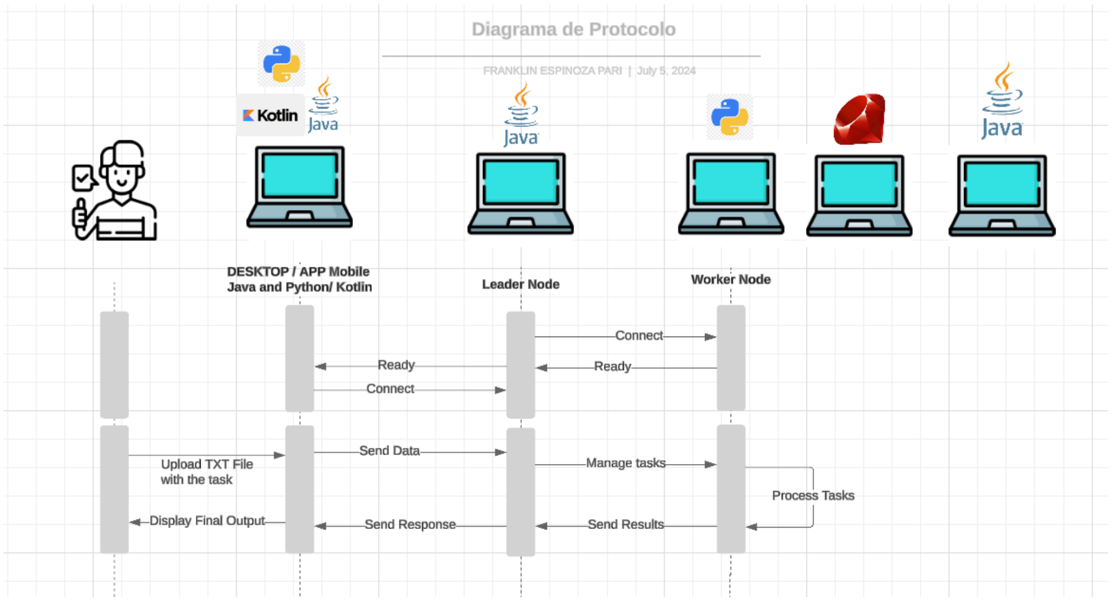
7. Diagram Architecture

The architecture of our distributed system is designed to handle tasks from various clients and process them using multiple worker nodes. The central component is the Leader Worker, implemented in **Java**, which coordinates the distribution and execution of tasks among different worker nodes. Clients can be developed in different programming languages, such as **Java**, **Kotlin**, and **Python**, and they send tasks to the Leader Worker. The worker nodes, which include implementations in **Ruby, Java** and **Python**, receive tasks from the Leader Worker, process them, and return the results. The system ensures fault tolerance by redistributing tasks if a node fails, and it allows for heterogeneous environments by supporting multiple operating systems and programming languages. Communication between clients, the Leader Worker, and worker nodes is managed through low-level sockets to maintain consistency and reliability in task execution. This design facilitates parallel and concurrent processing, enabling efficient handling of large-scale tasks across both LAN and Wi-Fi networks.



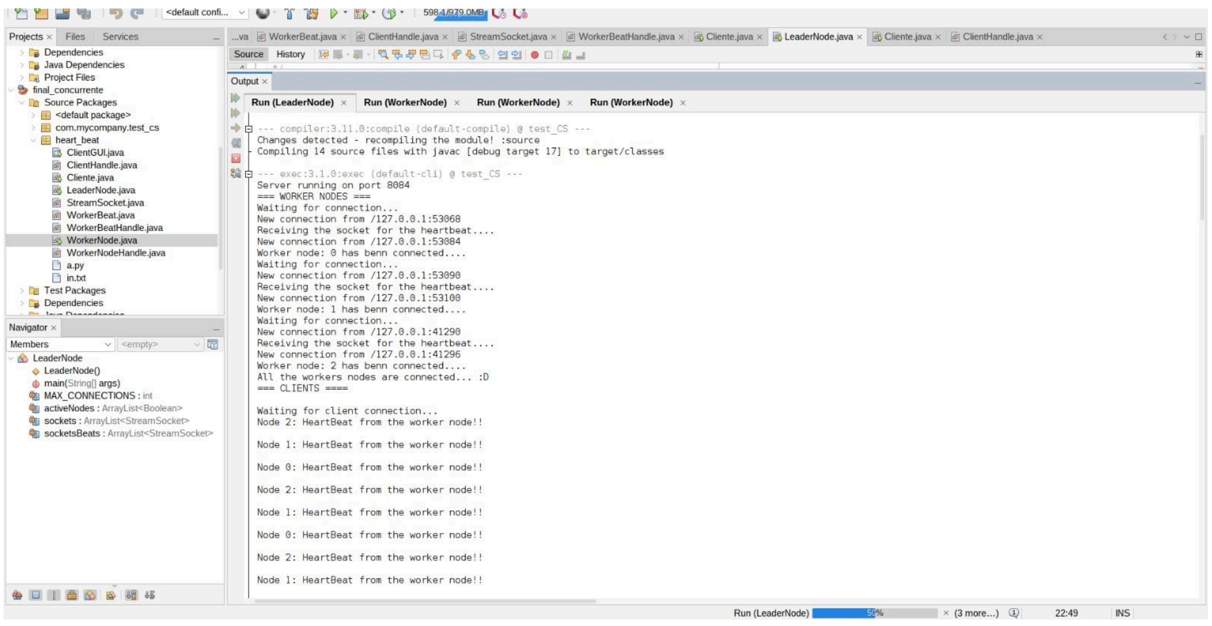
8. **Diagram Protoco**

The protocol diagram illustrates the interaction between the components of the distributed system. Clients, which can be desktop or mobile applications developed in Java, Kotlin, or Python, initiate the process by uploading a text file with the task to the Leader Node. The Leader Node, implemented in Java, coordinates the task distribution and communicates with various Worker Nodes. Upon connection, the Leader Node sends the task data to the Worker Nodes, which can be implemented in languages like Python, Ruby, and JavaScript. The Worker Nodes process the tasks and send the results back to the Leader Node. Finally, the Leader Node sends the results back to the clients, which display the final output. The diagram highlights the connectivity and readiness states of the nodes, the data transmission process, and the task management handled by the Leader Node to ensure efficient and fault-tolerant task processing in the distributed environment.



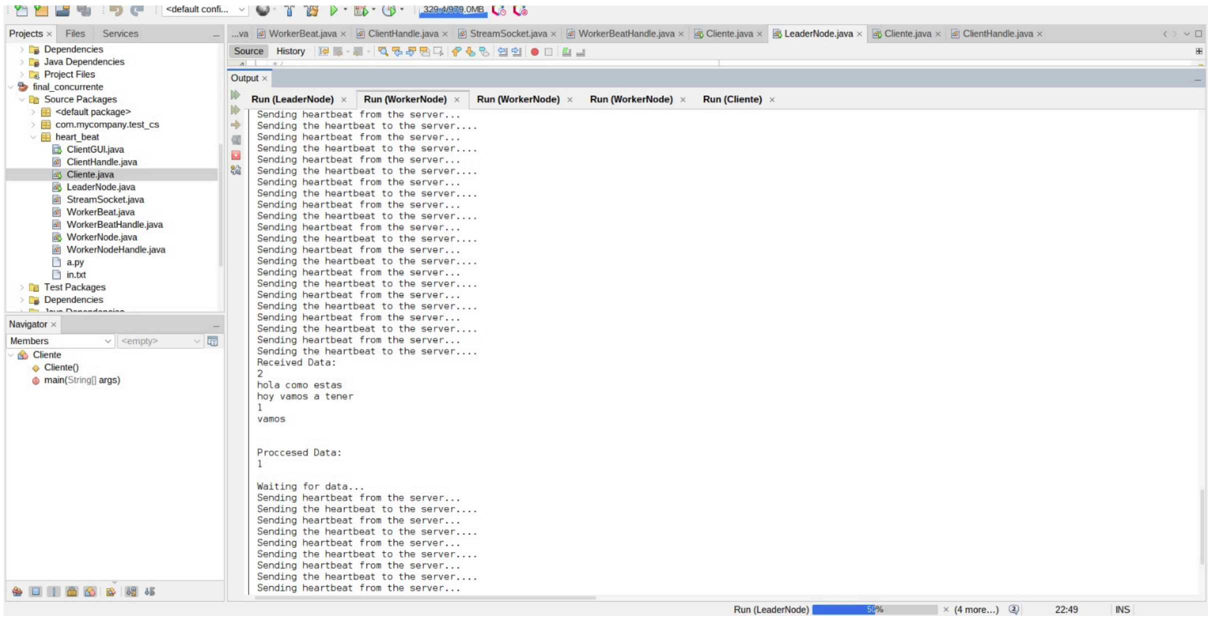
9. Development and Results

As indicated in the flowchart, we need to run the leader first.

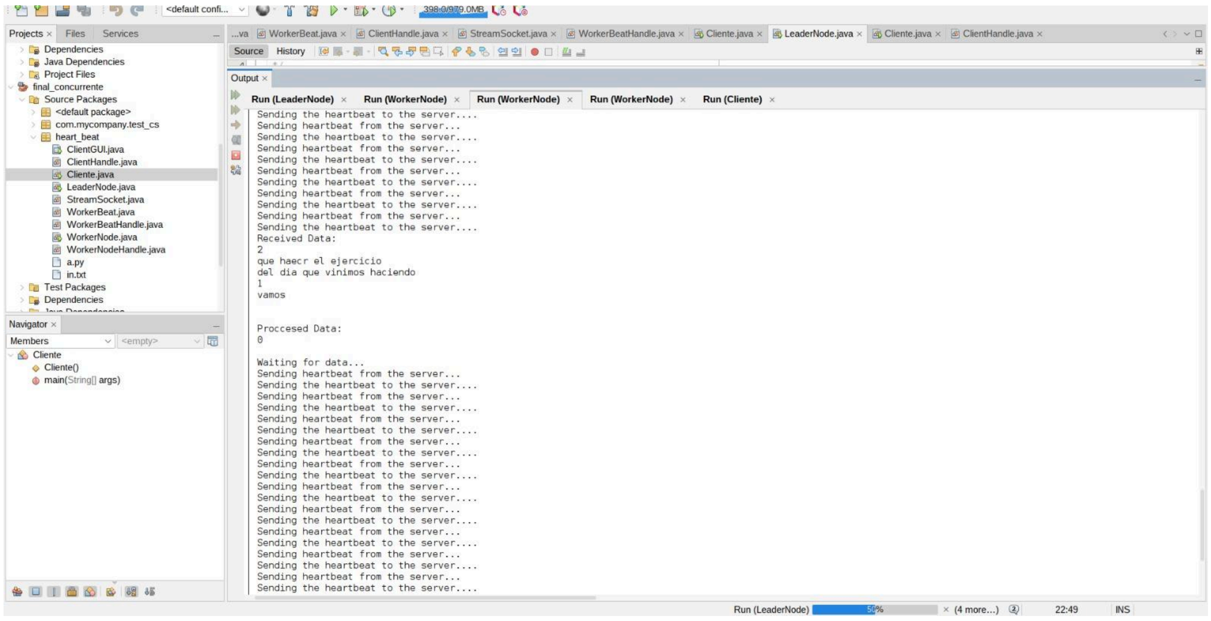


```
--- compiler:3.11.0:compile (default-compile) @ test_CS ---
Changes detected - recompiling the module! :source
Compiling 14 source files with javac [debug target 17] to target/classes

--- exec:3.1.0:exec (default-cli) @ test_CS ---
Server running on port 8084
=== WORKER NODES ===
Waiting for connection...
New connection from /127.0.0.1:53068
Receiving the socket for the heartbeat....
New connection from /127.0.0.1:53084
Worker node: 0 has benn connected....
Waiting for connection...
New connection from /127.0.0.1:53090
Receiving the socket for the heartbeat....
New connection from /127.0.0.1:53100
Worker node: 1 has benn connected....
Waiting for connection...
New connection from /127.0.0.1:41290
Receiving the socket for the heartbeat....
New connection from /127.0.0.1:41296
Worker node: 2 has benn connected....
All the workers nodes are connected... :D
=== CLIENTS ====

Waiting for client connection...
Node 2: HeartBeat from the worker node!!

Node 1: HeartBeat from the worker node!!

Node 0: HeartBeat from the worker node!!

Node 2: HeartBeat from the worker node!!

Node 1: HeartBeat from the worker node!!

Node 0: HeartBeat from the worker node!!

Node 2: HeartBeat from the worker node!!

Node 1: HeartBeat from the worker node!!
```

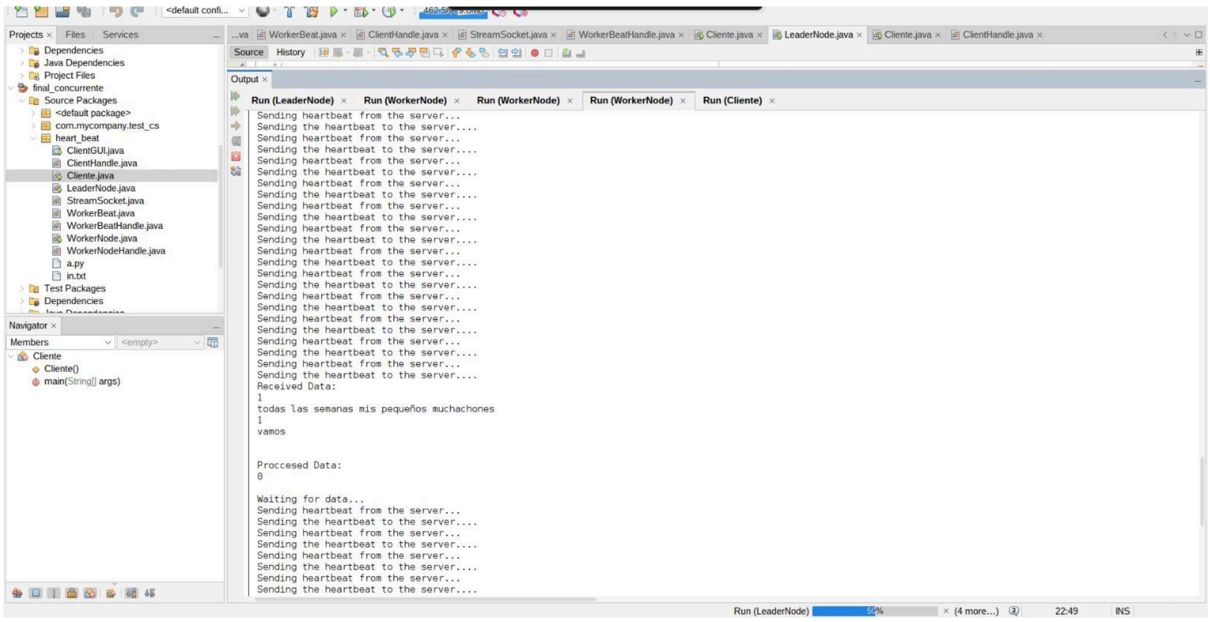
The workers now connect, in this case three workers will connect to the leader via sockets, specifying the leader's IP address and port.



```
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Received Data:
2
hola como estas
hoy vamos a tener
1
vamos


Proccesed Data:
1

Waiting for data...
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
```

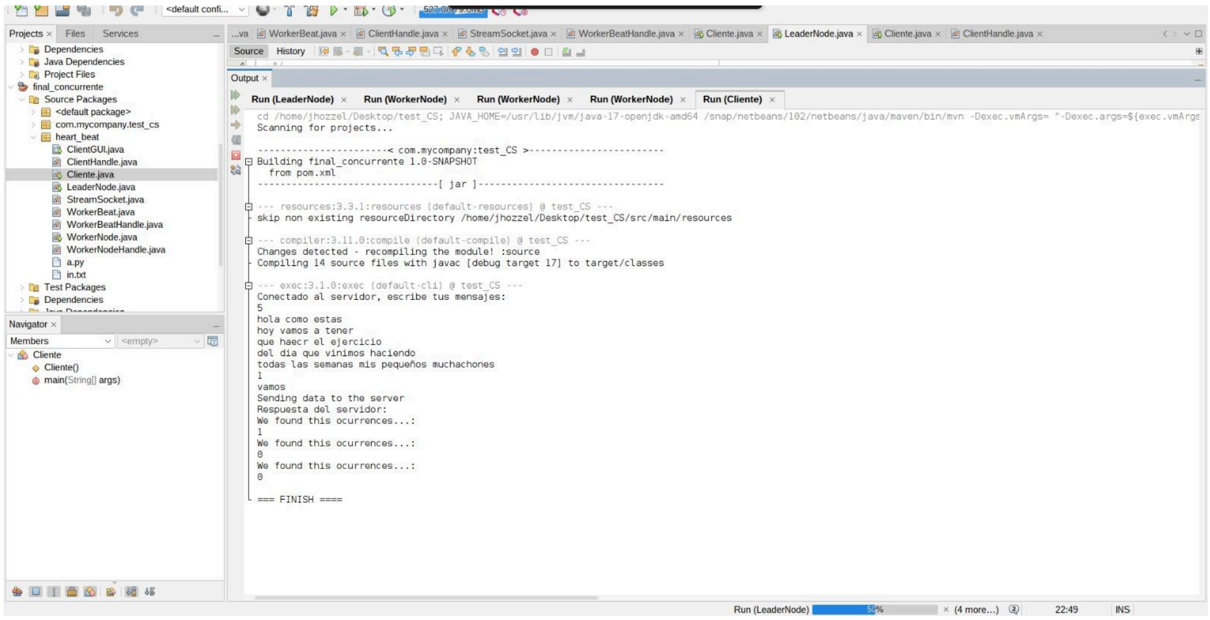


```
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Received Data:
2
que haecr el ejercicio
del dia que vinimos haciendo
1
vamos


Proccesed Data:
0

Waiting for data...
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
```



```
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Received Data:
1
todas las semanas mis pequeños muchachones
1
vamos


Proccesed Data:
0

Waiting for data...
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
Sending heartbeat from the server...
Sending the heartbeat to the server....
```

Then we need to run the client to connect to the leader worker to send requests.



```
cd /home/jhozzel/Desktop/test_CS; JAVA_HOME=/usr/lib/jvm/java-17-openjdk-amd64 /snap/netbeans/102/netbeans/java/maven/bin/mvn -Dexec.vmArgs= "-Dexec.args=${exec.vmArgs
Scanning for projects...

----------------------< com.mycompany:test_CS >-----------------------
Building final_concurrente 1.0-SNAPSHOT
  from pom.xml
--------------------------------[ jar ]--------------------------------

--- resources:3.3.1:resources (default-resources) @ test_CS ---
skip non existing resourceDirectory /home/jhozzel/Desktop/test_CS/src/main/resources

--- compiler:3.11.0:compile (default-compile) @ test_CS ---
Changes detected - recompiling the module! :source
Compiling 14 source files with javac [debug target 17] to target/classes

--- exec:3.1.0:exec (default-cli) @ test_CS ---
Conectado al servidor, escribe tus mensajes:
5
hola como estas
hoy vamos a tener
que haecr el ejercicio
del dia que vinimos haciendo
todas las semanas mis pequeños muchachones
1
vamos
Sending data to the server
Respuesta del servidor:
We found this ocurrences...:
1
We found this ocurrences...:
0
We found this ocurrences...:
0

=== FINISH ====
```

10. Conclusions

The report covers the comprehensive design, implementation, and evaluation of a distributed agent system for concurrent and parallel task processing. The system's architecture ensures fault tolerance, performance efficiency, and operational heterogeneity, making it suitable for diverse network environments.

11. Appendices

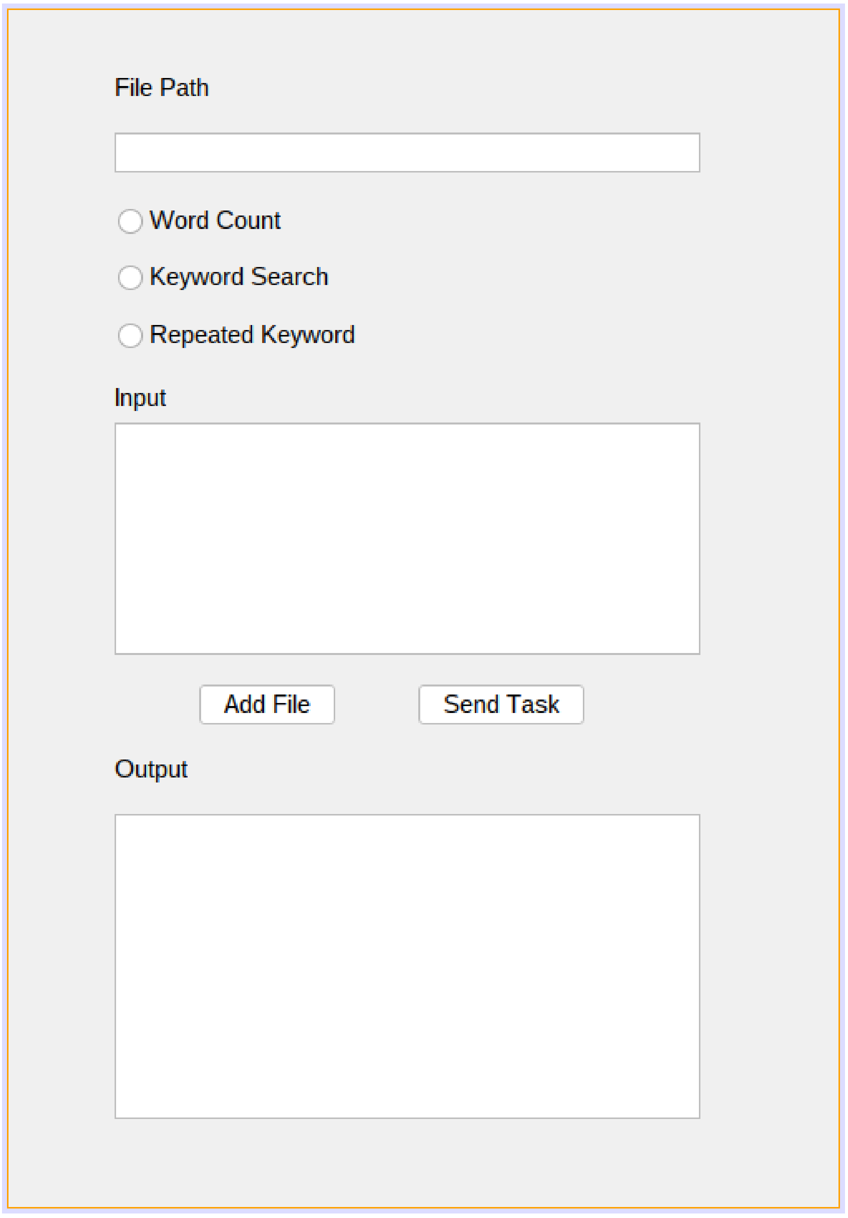
a. **Clients**

i. **Java Client**

```
package heart_beat;

import java.io.*;
import java.net.*;
import java.util.ArrayList;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Cliente {
        public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        try {
        int PUERTO = 8084;
        String ip = "10.10.0231";
        StreamSocket server = new StreamSocket(new Socket(ip, PUERTO));
        System.out.println("Conectado al servidor, texto mensajes:");
        BufferedReader consoleInput = new BufferedReader(new
InputStreamReader(System.in));

        String nombreArchivo =
"/home/cleber/Descargas/finalv11/test_CS/src/main/java/heart_beat/archivo.txt";
        File archivo = new File(nombreArchivo);
        int cantLinea = 0;

        while (true) {
                StringBuilder mssg = new StringBuilder();
                try {
                FileReader fr = new FileReader(archivo);
                BufferedReader br = new BufferedReader(fr);
                String linea;
                while ((linea = br.readLine()) != null) {
```

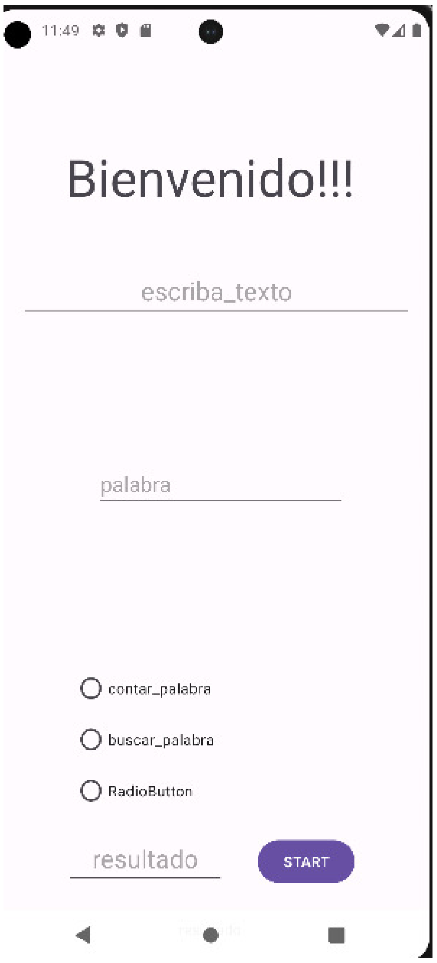
```
        cantLinea++;
        }
        System.out.println();
        fr = new FileReader(archivo);
        br = new BufferedReader(fr);
        String and = ""+cantLinea;
        mssg.append(and).append("\n");
        while ((linea = br.readLine()) != null) {
        mssg.append(linea).append('\n');
        }
        br.close();
        fr.close();
        } catch (IOException e) {
        System.err.println("Error al leer el archivo: " + e.getMessage());
        }
        System.out.print("Ingrese cuantas palabras quiere buscar: ");
        String nPalabras = consoleInput.readLine();
        mssg.append(nPalabras).append('\n');
        for(int i = 0; i < Integer.parseInt(nPalabras); i++){
        System.out.print("Ingrese la palabras "+i+" que quiere buscar: ");
        String line_ = in.nextLine();
        mssg.append(line_).append('\n');
        }
        System.out.println(mssg);
        System.out.println("Sending data to the server");
        server.send(mssg);

        // Reading data from the server
        mssg = server.receive();
        System.out.println("Respuesta del servidor: ");
        System.out.println(mssg.toString());
        System.out.println("=== FINISH ====");
    }
```

```
        } catch (IOException ex) {
        Logger.getLogger(Cliente.class.getName()).log(Level.SEVERE, null, ex);
        }
        }
}
```



ii. Kotlin Client

```
package com.concurrente.clienteraft;

import java.io.*;
import java.net.*;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Cliente {
    String texto;

    public Cliente(String texto) {
        this.texto = texto;
    }

    public void run() {
        int PUERTO = 8084;
        String ip = "10.10.0.231";
        Socket socket = null;
        PrintWriter out = null;
        BufferedReader in = null;

        try {
            socket = new Socket(ip, PUERTO);
            out = new PrintWriter(socket.getOutputStream(), true);
            in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

            // Enviar mensaje al servidor
            out.println(texto);

            // Recibir respuesta del servidor
            StringBuilder mssg = new StringBuilder();
            String line;
            while ((line = in.readLine()) != null) {
                mssg.append(line).append("\n");
            }

            System.out.println("Respuesta del servidor: ");
            System.out.println(mssg.toString());
            System.out.println("=== FINISH ====");
        } catch (UnknownHostException e) {
            Logger.getLogger(Cliente.class.getName()).log(Level.SEVERE, "Host desconocido: 
" + ip, e);
        } catch (IOException e) {
            Logger.getLogger(Cliente.class.getName()).log(Level.SEVERE, "Error de E/S al 
comunicarse con: " + ip, e);
        } finally {
            try {
                if (out != null) {
                    out.close();
                }
                if (in != null) {
                    in.close();
                }
                if (socket != null) {
                    socket.close();
                }
            } catch (IOException e) {
                Logger.getLogger(Cliente.class.getName()).log(Level.SEVERE, "Error al cerrar 
recursos", e);
            }
        }
    }

    public static void main(String[] args) {
        Cliente cliente = new Cliente("Tu mensaje aquí");
        cliente.run();
    }
}
```



iii. Python Client

```
import socket

class Cliente:
        def __init__(self):
        self.PUERTO = 8084
        self.ip = "10.10.0.231"
        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server_socket.connect((self.ip, self.PUERTO))
        print("Conectado al servidor, escribe tus mensajes:")

        def run(self):
        console_input = input
        while True:
        # Reading data from the console
        mssg = []
        n = int(console_input("Ingrese el número de elementos: "))
        mssg.append(str(n) + "\n")
        for i in range(n):
            line = console_input("Ingrese elemento {}: ".format(i+1))
            mssg.append(line + "\n")
        k = int(console_input("Ingrese el número de elementos adicionales: "))
        mssg.append(str(k) + "\n")
        for i in range(k):
            line = console_input("Ingrese elemento adicional {}: ".format(i+1))
            mssg.append(line + "\n")

        mssg_str = "".join(mssg)
        print("Enviando datos al servidor...")
        self.server_socket.sendall(mssg_str.encode())

        # Reading data from the server
        response = self.server_socket.recv(1024).decode()
        print("Respuesta del servidor:")
        print(response)
        print("=== FINISH ====")

if __name__ == "__main__":
    cliente = Cliente()
    cliente.run()
```

iv. Ruby Client

```
require 'socket'

class StreamSocket
  def initialize(socket)
        @socket = socket
  end
  def self.connect(ip, port)
        new(TCPSocket.new(ip, port))
  rescue StandardError => e
        puts "Connection error: #{e.message}"
        exit(1)
  end

  def receive
        @socket.gets
  rescue StandardError => e
        puts "Receive error: #{e.message}"
        exit(1)
  end

  def send(message)
        @socket.puts(message)
  rescue StandardError => e
        puts "Send error: #{e.message}"
        exit(1)
  end

  def close
        @socket.close
  end
end

class Cliente
  def self.run
        begin
        port = 8084
        ip = '10.10.0.231'
        server = StreamSocket.connect(ip, port)
        puts 'Conectado al servidor, escribe tus mensajes:'
```

```
          loop do
          # Leer datos de la consola
          print 'Ingresa el número de líneas: '
          n = gets.chomp.to_i
          mssg = ''

          n.times do
          line = gets.chomp
          mssg += line + "\n"
          end

          puts 'Sending data to the server'
          server.send(mssg)

          # Leer datos del servidor
          respuesta = server.receive
          puts 'Respuesta del servidor:'
          puts respuesta
          puts '=== FINISH ===='
          end
          rescue StandardError => e
          puts "Exception: #{e.message}"
          end
  end
end

Cliente.run
```

b. **Workers nodes**

i. **Java Worker - Leader**

```
package heart_beat;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.Socket;
import java.util.ArrayList;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;

public class WorkerNode {

        public static int f(String s) { // count the numbers of 'a'
        int cnt = 0;
        for (char c : s.toCharArray()) {
        if (c == 'a') cnt++;
        }
        return cnt;
        }

        public static void main(String[] args) {
        try {
        int PUERTO = 8084;
        String ip = "localhost";
        StreamSocket server = new StreamSocket(new Socket(ip, PUERTO));
        System.out.println("Connected to the leader node...");
        StreamSocket serverBeat = new StreamSocket(new Socket(ip, PUERTO));
        System.out.println("Creating the heartbeat channel to the server...");
        System.out.println("Starting the heartbeats...");

        new WorkerBeat(serverBeat).start();
        while (true) {
                System.out.println("Waiting for data...");
                StringBuilder mssg = server.receive();
                System.out.println("Received Data: ");
                System.out.println(mssg.toString());
                System.out.println("");

                // Process the data
                Scanner sc = new Scanner(mssg.toString());
                String line = sc.nextLine();
                int n = Integer.parseInt(line);
                ArrayList<String> a = new ArrayList<>();
                for (int i = 0; i < n; i++) {
                line = sc.nextLine();
                // Dividir la línea en palabras usando split()
                String[] palabras = line.split("\\s+"); // Esto divide por espacios en blanco

                // Agregar cada palabra al ArrayList
                for (String palabra : palabras) {
                a.add(palabra);
                }
                }
                n = a.size();

                line = sc.nextLine();
                int k = Integer.parseInt(line);
                ArrayList<String> b = new ArrayList<>();
                for (int i = 0; i < k; i++) {
                line = sc.nextLine();

                b.add(line);
                }
                int[] freq = new int [k];
                for (int i = 0; i < k; i++) {
                freq[i] = 0;
                }
                for (int i = 0; i < n; i++) {
                for (int j = 0; j < k; j++) {
```

```
                    if (b.get(j).equals(a.get(i))) {
                              freq[j]++;
                    }
                    }
                    }

                    StringBuilder ans = new StringBuilder();
                    for (int i = 0; i < k; i++) {
                    line = String.valueOf(freq[i]);
                    ans.append(line).append('\n');
                    }

                    System.out.println("Proccesed Data: ");
                    System.out.println(ans.toString());
                    server.send(ans);
          }
          } catch (IOException ex) {
          Logger.getLogger(Cliente.class.getName()).log(Level.SEVERE, null, ex);
          }
          }
}
```

—----------Heart_beat--

```
package heart_beat;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

class WorkerBeat extends Thread {
          private final StreamSocket server;
          //private final long SLEEP_TIME = 1000;  // Intervalo en milisegundos
```

```
        public WorkerBeat(StreamSocket server) {
        this.server = server;
        }

        @Override
        public void run() {
        try {
        //while (!Thread.currentThread().isInterrupted()) {  // Comprobar si el hilo no ha sido
interrumpido
        while (true) {
                StringBuilder sb = server.receive();
                System.out.println(sb.toString());
                System.out.println("Sending the heartbeat to the server....");
                server.send("HeartBeat from the worker node!!\n");
                //Thread.sleep(SLEEP_TIME);  // Dormir el hilo durante el intervalo
configurado
        }
        } catch (IOException e) {
        System.out.println("Error de entrada/salida en el HeartBeat: " + e.getMessage());
        /*} catch (InterruptedException e) {
        Thread.currentThread().interrupt();  // Restablecer el estado de interrupción
        System.out.println("HeartBeat interrumpido.");*/
        } finally {
        System.out.println("WorkerBeat finalizado.");
        }
        }
}
```

ii. Ruby Worker

```
require 'socket'
require 'thread'

class StreamSocket
```

```
  def initialize(socket)
          @socket = socket
  end

  def self.connect(ip, port)
          new(TCPSocket.new(ip, port))
  rescue StandardError => e
          puts "Connection error: #{e.message}"
          exit(1)
  end

  def receive
          @socket.gets
  rescue StandardError => e
          puts "Receive error: #{e.message}"
          exit(1)
  end

  def send(message)
          @socket.puts(message)
  rescue StandardError => e
          puts "Send error: #{e.message}"
          exit(1)
  end

  def close
          @socket.close
  end
end

class WorkerBeat
  def initialize(server)
          @server = server
  end
  def start
          Thread.new do
          loop do
          begin
          sb = @server.receive
          puts sb
          puts 'Sending the heartbeat to the server....'
          @server.send("HeartBeat from the worker node!!\n")
          sleep 1
          rescue StandardError => e
          puts "Heartbeat error: #{e.message}"
          break
          end
          end
          puts 'WorkerBeat finalizado.'
          end
  end
end

class WorkerNode
  def self.count_a(s)
          s.count('a')
  end

  def self.run
          begin
          port = 8084
          ip = '10.10.0.231'
          server = StreamSocket.connect(ip, port)
          puts 'Connected to the leader node...'
          server_beat = StreamSocket.connect(ip, port)
          puts 'Creating the heartbeat channel to the server...'
          puts 'Starting the heartbeats...'
```

```
            WorkerBeat.new(server_beat).start

            loop do
            puts 'Waiting for data...'
            msg = server.receive
            puts 'Received Data:'
            puts msg
            puts ''

            ans = ''
            msg.each_line do |line|
            next if line.strip.empty?

            puts "line: #{line}"
            line.split.each do |word|
            cnt = count_a(word)
            ans += "#{cnt}\n"
            end
            end

            puts 'Processed Data:'
            puts ans
            server.send(ans)
            end
            rescue StandardError => e
            puts "Exception: #{e.message}"
            end
  end
end

WorkerNode.run
```

12. References

- Tanenbaum, A. S., & Van Steen, M. (2007). Distributed Systems: Principles and Paradigms (2nd ed.). Pearson Education.

- Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified data processing on large clusters. In Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI '04) (pp. 137-150).

- Thain, D., Tannenbaum, T., & Livny, M. (2005). Distributed computing in practice: The Condor experience. Concurrency and Computation: Practice and Experience, 17(2-4), 323-356.

- Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2012). Distributed Systems: Concepts and Design (5th ed.). Addison-Wesley.